AI Orchestration Gateway

Independent project · 2026 · Solo · Private repository

One OpenAI-compatible endpoint in front of 14 AI providers that sends each request to one that can serve it and falls back automatically, with agents and a content studio on top.

Links: Repository (waske) is private; a walkthrough or read access is available on request.

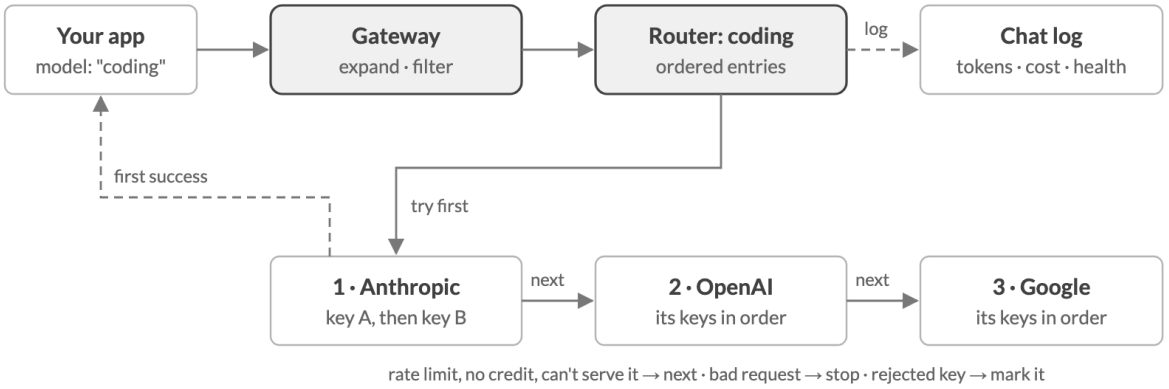
Why

Using several AI providers means several keys, rate limits, quotas and outages, and every app has to handle them. I wanted apps to call one endpoint with one model name, and leave the choice of provider and credential to a gateway that anyone can self-host without a managed database or queue.

What it does

- Exposes OpenAI-compatible chat, responses, images and videos APIs, plus Anthropic-shaped messages, behind API keys.
- Routers: a named fallback order across providers (for example "coding": Anthropic, then OpenAI, then Google), called like any model.
- Agents and souls: a soul is a reusable persona; an agent binds it to a router or model and is called by name. Editing a soul updates every agent that uses it.
- Studio: a question becomes a shot list, then images, then clips, then one finished video, all kept in a project.
- Several credentials per provider, tried in priority order, including Claude Code and Kimi Code subscription logins.
- Gates image, PDF, audio and video input by each model's capabilities, editable in the catalog.
- Per-request token usage and estimated cost, connection health, and a dashboard with admin and member roles.

Key flow



1. A client sends an OpenAI-format request; the model field is either a router name ("coding") or a direct "provider/model".
2. The gateway expands the router into one ordered list of candidates, entry by entry and connection by connection, and drops any that cannot serve this request.
3. It tries each candidate in turn. Every failure is classified: a rate limit moves to the next

candidate, a bad request stops at once, a rejected key marks that connection.
4. Each attempt is logged with tokens and estimated cost; the logs feed analytics and connection health.

Key decisions

Decision	Why / trade-off
OpenAI format as the default transport	Most providers copy it, so a new endpoint is a base URL, not a release. Only Anthropic, Google and Bedrock get their own client, because their protocol really differs.
Failures classified into a closed set of kinds	Fallback is only correct if the loop knows why an attempt failed. Each transport maps its errors to five kinds, so the judgement lives in one place instead of in every client.
Candidates expanded before the loop runs	All of a provider's keys are tried before the request moves to the next provider, which is what a user who put Anthropic first expects.
Agents are presets, not autonomous	An agent is a persona plus a router or model, resolved once before the fallback loop. No second execution path to maintain, and no tools, memory or chaining.
Circuit breaker kept in memory	Three failures open it for 30 seconds; a SQLite write per failure would cost more than the state is worth. It never empties the list, so a full outage still gets a try.
Secrets encrypted and unprintable	Stored keys are sealed with AES-256-GCM, and their type redacts itself, so a stray log line cannot leak someone's billable key.
One default-deny route table	The whole permission policy is one map. A new route that nobody classified requires admin, and a test fails until it is classified on purpose.

Non-goals

- A background worker for video jobs: the client polls for the result, and there is no webhook.

Outcome

- 180 merged pull requests in two months, with 31 architecture decision records explaining the choices.
- 1,359 backend tests using only the standard library and hand-written fakes; CI runs backend, frontend and a blocking Semgrep scan.
- Lesson: fallback is only as good as its error classification.

Stack

Go · Echo · SQLite · Next.js · React · TypeScript · AWS Bedrock · GitHub Actions · Semgrep